Title	Speed scaling with an arbitrary power function
Author(s)	Bansal, N; Chan, HL; Pruhs, K
Citation	The 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), New York, N.Y., 4-6 January 2009. In Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, 2009, p. 693-701
Issued Date	2009
URL	http://hdl.handle.net/10722/125690
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Speed Scaling with an Arbitrary Power Function

Nikhil Bansal* Ho-Leung Chan† Kirk Pruhs‡

"What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city."
—Dr. Eric Schmidt, CEO of Google [12].

Abstract

All of the theoretical speed scaling research to date has assumed that the power function, which expresses the power consumption P as a function of the processor speed s, is of the form $P = s^\alpha$, where $\alpha > 1$ is some constant. Motivated in part by technological advances, we initiate a study of speed scaling with arbitrary power functions. We consider the problem of minimizing the total flow plus energy. Our main result is a $(3+\epsilon)$-competitive algorithm for this problem, that holds for essentially any power function. We also give a $(2+\epsilon)$-competitive algorithm for the objective of fractional weighted flow plus energy. Even for power functions of the form s^α, it was not previously known how to obtain competitiveness independent of α for these problems. We also introduce a model of allowable speeds that generalizes all known models in the literature.

1 Introduction

Energy consumption has become a key issue in the design of microprocessors. Major chip manufacturers, such as Intel, AMD and IBM, now produce chips with dynamically scalable speeds, and produce associated software that enables an operating system to manage power by scaling processor speed. Within the last few years there has been a significant amount of research on the scheduling problems that arise in this setting. Generally these problems have dual objectives as one wants both to optimize some schedule quality of service objective (for example, total flow) and some power related objective (for example, the total energy used). Scheduling algorithms for these problems have two components: A *job selection* policy that determines which job to run and a *speed scaling* policy to determine the speed at which the processor is run.

All of the theoretical speed scaling research to date has assumed that the power function, which expresses the power consumption P as a function of the processor speed s, is of the form $P = s^\alpha$, where $\alpha > 1$ is some constant. Let us call this the *traditional model*. The traditional model was motived by the fact that in CMOS based processors, the well known cube-root rule states that the speed is approximately the cube root of the power. So historically $P = s^3$ was a reasonable assumption. In the literature one finds different variations on this traditional model based on which speeds are allowable. Most of the literature assumes the *unbounded speed model*, in which a processor can be run at any real speed in the range $[0, \infty)$. Some of the literature assumes the *bounded speed model* in which the allowable speeds lie in some real interval $[0, T]$. Some of the literature on offline algorithms, assumes the *discrete speeds model* in which there are a finite number of allowable speeds.

Our main contribution in this paper is to initiate theoretical investigations into speed scaling problems with more general power functions, and develop algorithmic analysis techniques for this setting. For an explanation of the historical technological motivation for the traditional model, and the current technological motivations for considering more general power functions, see section 1.3. A secondary contribution is to introduce a model for allowable speeds that generalizes all of various models found in the literature.

We will consider the objective of minimizing a linear combination of total (possibly weighted) flow and total energy used. Our third contribution is to improve on the known results for this important fundamental problem. Optimizing a linear combination of energy and total flow has the following natural interpretation. Suppose that the user specifies how much improvement in flow, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify that he is willing to spend 1 erg of energy from the battery for a decrease of 4 micro-seconds in flow. Then the optimal schedule, from this user's perspective, is the schedule

*IBM T.J. Watson Research, P.O. Box 218, Yorktown Heights, NY. nikhil@us.ibm.com

†Max-Planck-Institut für Informatik. hlchan@mpi-inf.mpg.de. This paper was done when the author was in University of Pittsburgh.

‡Computer Science Department, University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by an IBM faculty award, and by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558.

that optimizes $\rho = 4$ times the energy used plus the total flow. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$. Weighted flow generalizes both total flow, and total/average stretch, which is another common QoS measure. The stretch/slowdown of a job is the flow divided by the work of the job. When the user is aware of the size of a job (say if the user knows that he/she is downloading a video file instead of a text file) then perhaps slowdown is a more appropriate measure of the happiness of a user than flow. Many server systems, such as operating systems and databases, have mechanisms that allow the user or the system to give different priorities to different jobs. For example, Unix has the `nice` command. In a speed scaling setting, the weight of a job is indicative of the flow versus energy trade-off for this job. The user may be willing to spend more energy to reduce the flow of a higher priority job, than for a lower priority job.

1.1 The Literature on Flow + Energy in the Traditional Model Let us start with results in the unbounded speed model. Pruhs, Uthaisombut and Woeginger [15] gave an efficient offline algorithm to find the schedule that minimizes average flow subject to a constraint on the amount of energy used, in the case that jobs have unit work. This algorithm can also be used to find optimal schedules when the objective is a linear combination of total flow and energy used. They observed that in any locally-optimal schedule, essentially each job i is run at a power proportional to the number of jobs that would be delayed if job i was delayed. Albers and Fujiwara [1] proposed the natural online speed scaling algorithm that always runs at a power equal to the number of unfinished jobs (which is lower bound to the number of jobs that would be delayed if the selected job was delayed). They did not actually analyze this natural algorithm, but rather analyzed a batched variation, in which jobs that are released while the current batch is running are ignored until the current batch finishes. They showed that for unit work jobs that this batched algorithm is $O\left(\left(\frac{3+\sqrt{5}}{2}\right)^{\alpha}\right)$-competitive by reasoning directly about the optimal schedule. This gave a competitive ratio of about 400 when the cube-root rule holds. They also gave an efficient offline dynamic programming algorithm. Bansal, Pruhs and Stein [4] considered the algorithm that runs at a power equal to the unfinished work (which is in general a bit less than the number of unfinished jobs for unit work jobs). They showed that for unit work jobs, this algorithm is 2-competitive with respect to the objective of fractional flow plus energy using an amortized local competitiveness argument. A job that is say 2/3 completed at time t, only contributes 1/3 to the increase in fractional flow at time t. They then showed that the natural algorithm proposed in [1] is 4-competitive for total flow plus energy for unit work jobs.

For the more general setting where jobs have arbitrary sizes and arbitrary weights and the objective is weighted flow plus energy, Bansal, Pruhs and Stein [4] considered the algorithm that uses Highest Density First (HDF) for job selection, and always runs at a power equal to the fractional weight of the unfinished jobs. They showed that this algorithm is $O(\frac{\alpha}{\log \alpha})$-competitive for fractional weighted flow plus energy using an amortized local competitiveness argument. Using the known resource augmentation analysis of HDF [5], they then showed how to obtain an $O(\frac{\alpha^2}{\log^2 \alpha})$-competitive algorithm for (integral) weighted flow plus energy. The competitive ratio was a bit less than 8 when the cube-root rule holds.

Recently, Lam et al. [11] improved the competitive ratio for total flow plus energy for arbitrary work and unit weight jobs. They considered the job selection algorithm Shortest Remaining Processing Time (SRPT) and the speed scaling algorithm of running at a power proportional to the number of unfinished jobs, and proved that this is $O(\frac{\alpha}{\log \alpha})$-competitive for flow time plus energy. When the cube-root rule holds, this competitive ratio was about 3.25. Their improvement came by reasoning directly about integral flow time instead of arguing first about fractional flow time. Speed scaling papers prior to [11] used potential functions related to the fractional amount of unfinished work or weight. The main reason for this was that such a potential function varies continuously with time which substantially simplifies the proofs as one only needs to consider instantaneous states of the online and offline algorithms. The main technical contribution of [11] was the introduction of a different form of continuously varying potential function that depends on the integral number of unfinished jobs.

Bansal et al. [2] extended the results of [4] for the unbounded speed model to the bounded speed model. The speed scaling algorithm was to run at the minimum of the speed recommended by the speed scaling algorithm in the unbounded speed model and the maximum speed of the processor. The contribution there was to develop the algorithmic analysis techniques necessary to analyze this algorithm. The results for the bounded speed model in [2] were improved in [11], again by reasoning directly about integral flow. Again [11] showed competitive ratios of $O(\frac{\alpha}{\log \alpha})$. When the cube-root rule holds, the obtained competitive ratio was 4.

1.2 Our Results We assume that the allowable speeds are a countable collection of disjoint subintervals of $[0, \infty)$. We assume that all the intervals, except possibly the rightmost interval, are closed on both ends. The rightmost interval may be open on the right if the power $P(s)$ approaches infinity as the speed s approaches the rightmost endpoint of that interval. We assume that P is non-negative, and P is continuous and differentiable on all but countably many points. We assume that either there is a maximum allowable speed T, or that the limit inferior of $P(s)/s$ as s approaches infinity is not zero (if this condition doesn't hold then, then the optimal speed scaling policy is to run at infinite speed). Let us call this the *general model*. We give two main results in the general model.

THEOREM 1.1. *Consider the scheduling algorithm that uses Shortest Remaining Processing Time (SRPT) for job selection and power equal to one more than the number of unfinished jobs for speed scaling. In the general model, this scheduling algorithm is* $(3 + \epsilon)$*-competitive for the objective of total flow plus energy on arbitrary-work unit-weight jobs.*

THEOREM 1.2. *Consider the scheduling algorithm that uses Highest Density First (HDF) for job selection and power equal to the fractional weight of the unfinished jobs for speed scaling. In the general model, this scheduling algorithm is* $(2 + \epsilon)$*-competitive for the objective of fractional weighted flow plus energy on arbitrary-work arbitrary-weight jobs.*

We establish these results through an amortized local competitiveness argument. As in [2], our potential function is based on the integral number of unfinished jobs. However, our potential function is quite different and more general than the potential function considered in [2]. While Theorem 1.1 deals with integral flow, Theorem 1.2 only holds for fractional weighted flow. Obtaining a competitive ratio independent of α for the objective of (integral) weighted flow plus energy is ruled out since resource augmentation is required to achieve $O(1)$-competitiveness for the objective of weighted flow on a fixed speed processor [3].

Let us consider what these theorems say in the traditional model. Theorem 1.1 slightly improves the best known competitive ratios, when the cube root rule holds, in both the unbounded and bounded speed models. The potential functions used in all previous papers are specifically tailored toward the function $P = s^\alpha$. Moreover, these potential functions can not be used to show competitive ratios of $o(\frac{\alpha}{\ln \alpha})$ for arbitrary work jobs. So while the competitive ratios were $O(1)$-competitive for a fixed α, they were not $O(1)$-competitive for a general power function of the form s^α. Our new potential function not only allows us to break the barrier of $\frac{\alpha}{\ln \alpha}$, and it allows us to obtain $O(1)$-competitiveness for general power functions.

1.3 Technological Motivations The power used by a processor can be partitioned into dynamic power, the power used by switching when doing computations, and static/leakage power, which is the power lost in absence of any switching. Historically (say up until 5 years ago) the static power used by a processor was negligible compared to the dynamic power. Dynamic power is roughly proportional to sV^2, where V is the voltage. However as the minimum voltage required to drive the microprocessor at a desired speed is approximately proportional to the frequency [7], this leads to the well known cube-root rule that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P = s^3$, the power is proportional to the speed cubed [7].

However, currently the static power used by the common processors manufactured by Intel and AMD is now comparable to the dynamic power. The reason for this is that to increase processor speeds, transistor sizes and desired transistor switching delays must decrease. To obtain a smaller switching delay, the threshold voltage for the transistor must be decreased. Unfortunately, subthreshold leakage current increases exponentially as threshold voltage decreases [13, 8]. This suggests modeling the speed to power function as something like $P(s) = s^\alpha + c$, where there is some range of speeds $[s_{min}, s_{max}]$ that the processor can run at. Here $\alpha \approx 3$ comes from the cube-root rule for dynamic power, and c is a constant specifying the static power loss. There is however, good motivation for considering more general speed to power functions. We will state three more examples here.

Example 1: In general, the higher speed that one can scale a core or a processor, the greater the static power (because the threshold voltage must be less). It has been proposed that it might be advantageous to build a processor consisting of different circuitry for running jobs at different speeds speed ranges [6, 8]. Each circuitry would have different speed ranges, different static powers due to the different threshold voltages, and different power functions. Thus the speed scaling algorithm might be able to choose among a collection of power functions of the form $P_i(s) = a_i s^\alpha + c_i$, where each $P_i(s)$ is applicable over a collection $S_i = \{s_{i,1}, \ldots, s_{i,k_i}\}$ of speeds. (Note that in this context that we can not scale our units so that the multiplicative constant is always 1). In this setting, the power function of might be of the form:

$$P(s) = \min_{i: s \in S_i} a_i s^\alpha + c_i$$

Example 2: Subthreshold leakage current/power is not independent of temperature. The temperature in turn depends on the speed that the processor is run. Some discussion of how leakage current/power is related to temperature can be found in [8], but it seems that this issue is perhaps not so well understood. But in any case, there appears to be no reason to presume that the resulting power function would be of the form s^α.

Example 3: Another motivation for considering general speed to power functions is at the data center level. The opening quote indicates the importance of power management at the data center level. A nice investigation, by Google researchers, into the energy that a data center could save from speed scaling can be found in [9]. Ultimately what a data center operator might care about is the cost of the energy used, not the actual amount of energy used. These can be quite different because normally the contract that the data center has with the electrical suppliers can have drastic increases in cost if various power thresholds are exceeded. So here the relevant function that one would care about is the speed to cost (say measured in dollars per second) function, not the speed to power function. There is no reason why these contracts need to have a cost that rises as s^α.

2 Preliminaries

An instance consists of n jobs, where job i has a release time r_i, and a positive work y_i. In some cases each job may have a positive integer weight w_i. An online scheduler is not aware of job i until time r_i, and, at time r_i, learns y_i and weight w_i. For each time, a scheduler specifies a job to be run and a speed at which the processor is run. We assume that preemption is allowed, that is, a job may be suspended and later restarted from the point of suspension. A job i completes once y_i units of work have been performed on i. The speed is the rate at which work is completed; a job with work y run at a constant speed s completes in $\frac{y}{s}$ seconds. The flow F_i of a job i is its completion time minus its release time. The total flow is $\sum_{i=1}^n F_i$. The weighted flow is $\sum_{i=1}^n w_i \cdot F_i$.

Let us quickly review amortized local competitiveness analysis. Consider an objective G. Let $G_A(t)$ be the increase in the objective in the schedule for algorithm A at time t. So when G is total flow plus energy, $G_A(t)$ is $P(s_a^t) + n_a^t$, where s_a^t is the speed for A at time t and n_a^t is the number of unfinished jobs for A at time t. This is because energy is power integrated over time, and total flow is the number of unfinished jobs integrated over time. Let OPT be the offline adversary that optimizes G and A be some algorithm. To prove A is c-competitive it suffices to give a potential function $\Phi(t)$ such that the following two conditions hold.

Boundary condition: Φ is zero before any job is released and Φ is non-negative after all jobs are finished.

General condition: At any time t,

$$G_A(t) + \frac{d\Phi(t)}{dt} \leq c \cdot G_{OPT}(t) \tag{2.1}$$

To prove the general condition, it suffices to show that Φ does not increase when a job arrives or is completed by A or OPT, and Equation 2.1 is true during any period of time without job arrival or completion. By integrating Equation 2.1, we can see that A is c-competitive for H. For more information see [14].

3 Flow Plus Energy

Our goal in this section is to prove Theorem 1.1, that in the general model SRPT plus the natural speed scaling algorithm is $(3+\epsilon)$-competitive for the objective of total flow plus energy. We start by showing that this algorithm is 3-competitive if P also satisfies the following additional *special conditions*:

- P is defined, and continuous and differentiable at all speeds in $[0, \infty)$.
- $P(0) = 0$.
- P is strictly increasing.
- P is strictly convex. That is, for $a < b$ and for all $p \in (0,1)$, is the case that $pP(a) + (1-p)P(b) > P(pa + (1-p)b)$.
- P is unbounded. That is, for all c, the exists a speed s such that $P(s) > c$.

Then in section 3.1 we show how to extend the analysis to the case when these special conditions do not hold, at the cost of an arbitrarily small increase in the competitive ratio.

Definition of online algorithm A: Algorithm A schedules the unfinished job with the least remaining unfinished work, and runs at speed s_a^t where

$$s_a^t = \begin{cases} P^{-1}(n_a^t + 1) & \text{if } n_a^t \geq 1 \\ 0 & \text{if } n_a^t = 0 \end{cases}$$

where n_a^t is the number of unfinished jobs for A at time t. As $P(0) = 0$, and P is strictly increasing, continuous and unbounded, $P^{-1}(i)$ exists and is unique for all integers $i \geq 1$. Thus, A is well defined.

Definition of potential function Φ: Let OPT be the offline adversary that minimizes total flow plus energy. We can assume without loss of generality that

OPT runs SRPT. At any time t, let n_o^t be the number of unfinished jobs in OPT. Let $n_a^t(q)$ and $n_o^t(q)$ be the number of unfinished jobs with remaining size at least q in A and OPT, respectively, at time t. Let $n^t(q) = \max\{0, n_a^t(q) - n_o^t(q)\}$. We define the the potential function

$$\Phi(t) = 3\int_0^\infty f(n^t(q))dq$$

where f is defined by

$$\begin{aligned} f(0) &= 0 \\ \forall i \geq 1,\ f(i) - f(i-1) &= P'(P^{-1}(i)) \end{aligned}$$

and $P'(x)$ is the derivative of $P(x)$. Note that $P'(P^{-1}(i))$ simply means substituting $x = P^{-1}(i)$ into $P'(x)$. Since $P(x)$ is differentiable, $P'(x)$ is well-defined. Hence, f is well-defined for all integer i. As both $P'(x)$ and $P^{-1}(i)$ are increasing, we have $f(i) - f(i-1) \geq f(j) - f(j-1)$ if $i > j$. We denote $\Delta(i) = f(i) - f(i-1)$ for simplicity.

We now wish to establish a crucial technical lemma, Lemma 3.1, which relies on the well known Young's inequality.

THEOREM 3.1. (YOUNG'S INEQUALITY[10]) *Let g be a real-valued, continuous, and strictly increasing function on $[0, c]$ with $c > 0$. If $g(0) \geq 0$, and a, b such that $a \in [0, c]$, and $b \in [g(0), g(c)]$, then*

$$\int_0^a g(x)dx + \int_{g(0)}^b g^{-1}(x)dx \geq ab$$

where g^{-1} is the inverse function of g.

LEMMA 3.1. *Let P be a strictly increasing, strictly convex, continuous and differentiable function. Let $i, s_a, s_o \geq 0$ be any real. Then*

$$\begin{aligned} &P'(P^{-1}(i))(-s_a + s_o) \\ &\quad \leq \ (-s_a + P^{-1}(i))P'(P^{-1}(i)) + P(s_o) - i \end{aligned}$$

Proof. Since P is strictly increasing and strictly convex, $P'(0) \geq 0$ and $P'(x)$ is strictly increasing. Thus, by Young's Inequality with $g(x) = P'(x)$, $a = s_o$ and $b = P'(P^{-1}(i))$, we have

$$\begin{aligned} & s_o P'(P^{-1}(i)) \\ \leq\ & \int_0^{s_o} g(x)dx + \int_{P'(0)}^{P'(P^{-1}(i))} g^{-1}(x)dx \\ =\ & P(s_o) + \left[xg^{-1}(x)\right]_{g(0)}^{g(P^{-1}(i))} - \int_{g(0)}^{g(P^{-1}(i))} x d(g^{-1}(x)) \\ =\ & P(s_o) + g(P^{-1}(w))P^{-1}(i) - \int_0^{P^{-1}(i)} g(y)dy \\ =\ & P(s_o) + g(P^{-1}(i))P^{-1}(i) - i \end{aligned}$$

The first equality is obtained by integration by parts and the second equality is obtained by letting $y = g^{-1}(x)$. The lemma follows by adding $-s_a P'(P^{-1}(i))$ to both sides.

We are now ready to proceed with our amortized local competitiveness argument. For the boundary condition, we observe that before any job is released and after all jobs are finished, $n^t(q) = 0$ for all q, so $\Phi = 0$. For the general condition, we observe that when a job is released, $n^t(q)$ is not changed for all q, so Φ is unchanged. When a job is completed by A or OPT, $n^t(q)$ is changed only at the single point of $q = 0$, which does not affect the integration, so Φ is also unchanged. It remains to show at any time t during a time interval without job arrival or completion,

$$n_a^t + P(s_a^t) + \frac{d}{dt}\Phi(t) \leq 3(n_o^t + P(s_o^t)) \tag{3.2}$$

The rest of this proof considers any such time t and proves Equation 3.2. We omit t from the superscript and the parameter for convenience. First observe that if $n_a = 0$, then $n(q) = 0$ for all q, and $\frac{d}{dt}\Phi = 0$. Thus equation 3.2 trivially holds. Henceforth, we assume $n_a \geq 1$.

Let q_a be the remaining size of the job with the shortest remaining size in A (q_a exists as $n_a \geq 1$). Note that A is processing a job of size q_a. Define q_o similarly for OPT ($q_o = 0$ if $n_o = 0$). Consider an infinitesimal interval of time $[t, t + dt]$. The processing of A causes $n_a(q)$ to decrease by 1 for $q \in [q_a - s_a dt, q_a]$. Similarly, the processing of OPT causes $n_o(q)$ to decrease by 1 for $q \in [q_o - s_o dt, q_o]$. Denote the change in Φ during this time interval as $d\Phi$. We consider three cases depending on whether $n_o > n_a$, $n_o < n_a$, or $n_o = n_a$.

Case $n_o > n_a$: We show that $d\Phi \leq 0$, as follows. At time t, $n_o(q)$ is greater than $n_a(q)$ for q in $[q_o - s_o dt, q_o]$. Thus, at time $t + dt$, $n_o(q)$ is still at least $n_a(q)$ for q in $[q_o - s_o dt, q_o]$. It means that $n(q)$ remains zero in this interval and Φ is not increased due to the processing of OPT. The processing of A only decreases Φ or leaves Φ unchanged. Hence, $d\Phi \leq 0$. Thus implies that $\frac{d}{dt}\Phi \leq 0$ and $n_a + P(s_a) + \frac{d}{dt}\Phi \leq n_a + n_a + 1 \leq 3n_o$, so (3.2) holds.

Case $n_o < n_a$: We show that either $d\Phi \leq 3\Delta(n_a - n_o)(-s_a + s_o)dt$ or $d\Phi \leq 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$, as follows. Consider 3 subcases depending on whether q_a is smaller than, equal to, or bigger than q_o.

1. Assume $q_a < q_o$. As $n_a(q)$ decreases by 1 for q in $[q_a - s_a dt, q_a]$, by the definition of Φ, the change in Φ due to A is $3(f(n_a(q_a) - n_o(q_a) - 1) - f(n_a(q_a) - n_o(q_a)))s_a dt = -3\Delta(n_a(q_a) - n_o(q_a))s_a dt$. Since

$q_a < q_o$, we have $n_o(q_a) = n_o(q_o) = n_o$. So the change in Φ due to A is $-3\Delta(n_a - n_o)s_a dt$.

As $n_o(q)$ decreases by 1 for q in $[q_o - s_o dt, q_o]$, by the definition of Φ, the change in Φ due to OPT is at most $3(f(n_a(q_o) - n_o(q_o) + 1) - f(n_a(q_o) - n_o(q_o)))s_o dt = 3\Delta(n_a(q_o) - n_o(q_o) + 1)s_o dt$. Since $q_a < q_o$, we have $n_a(q_o) \leq n_a(q_a) - 1$. Thus, $n_a(q_o) - n_o(q_o) + 1 \leq n_a(q_a) - n_o(q_o)$. As $\Delta(i) \leq \Delta(j)$ for $i \leq j$, the change in Φ due to OPT is at most $3\Delta(n_a(q_a) - n_o(q_o))s_o dt = 3\Delta(n_a - n_o)s_o dt$. Adding up the change in Φ due to A and OPT, we obtain that $d\Phi \leq 3\Delta(n_a - n_o)(-s_a + s_o)dt$.

2. Assume $q_a = q_o$. If $s_a \geq s_o$, $n(q) = n_a(q) - n_o(q)$ decreases by 1 for q in $[q_a - s_a dt, q_a - s_o dt]$. Hence, the change in Φ is $3(f(n_a(q_a) - n_o(q_a) - 1) - f(n_a(q_a) - n_o(q_a)))(s_a - s_o)dt = 3\Delta(n_a - n_o)(-s_a + s_o)dt$. Else if $s_a < s_o$, $n(q) = n_a(q) - n_o(q)$ increases by 1 for q in $[q_a - s_o dt, q_a - s_a dt]$. Hence, the change in Φ is $3(f(n_a(q_a) - n_o(q_a) + 1) - f(n_a(q_a) - n_o(q_a)))(s_o - s_a)dt = 3\Delta(n_a - n_0 + 1)(-s_a + s_o)dt$.

3. Assume $q_a > q_o$. As $n_a(q)$ decreases by 1 for q in $[q_a - s_a dt, q_a]$, the change in Φ due to A is $3(f(n_a(q_a) - n_o(q_a) - 1) - f(n_a(q_a) - n_o(q_a)))s_a dt = -3\Delta(n_a(q_a) - n_o(q_a))s_a dt$. Since $q_a > q_o$, we have $n_o(q_a) \leq n_o(q_o) - 1$. Thus, $n_a(q_a) - n_o(q_a) \geq n_a(q_a) - n_o(q_o) + 1$. As $-\Delta(i) \leq -\Delta(j)$ for $i \geq j$, The change in Φ due to A is at most $-3\Delta(n_a(q_a) - n_o(q_o) + 1)s_a dt = -3\Delta(n_a - n_o + 1)s_a dt$.

   On the other hand, $n_o(q)$ decreases by 1 for q in $[q_o - s_o dt, q_o]$, so the change in Φ due to OPT is at most $3(f(n_a(q_o) - n_o(q_o) + 1) - f(n_a(q_o) - n_o(q_o)))s_o dt = 3\Delta(n_a(q_o) - n_o(q_o) + 1)s_o dt$. Since $q_a > q_o$, $n_a(q_o) = n_a(q_a) = n_a$, and the change in Φ due to OPT is at most $3\Delta(n_a - n_o + 1)s_o dt$.

   Summing the change in Φ due to A and OPT, $d\Phi \leq 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$.

Combining the above 3 subcases, it implies that if $n_o < n_a$, then either

$$\begin{aligned} \frac{d}{dt}\Phi &\leq 3\Delta(n_a - n_0)(-s_a + s_o) \quad \text{or} \\ \frac{d}{dt}\Phi &\leq 3\Delta(n_a - n_0 + 1)(-s_a + s_o). \end{aligned}$$

Note that $\Delta(n_a - n_o)(-s_a + s_o) = P'(P^{-1}(n_a - n_o))(-s_a + s_o)$. Setting $i = n_a - n_o$ in Lemma 3.1, we have that

$$\begin{aligned} & P'(P^{-1}(n_a - n_o))(-s_a + s_o) \\ \leq\ & (-s_a + P^{-1}(n_a - n_o))P'(P^{-1}(n_a - n_o)) + \\ & \quad +P(s_o) - n_a + n_o \\ \leq\ & P(s_o) - n_a + n_o, \end{aligned}$$

where the last inequality follows from $s_a = P^{-1}(n_a + 1) \geq P^{-1}(n_a - n_o)$. Similarly, $\Delta(n_a - n_o + 1)(-s_a + s_o) = P'(P^{-1}(n_a - n_o + 1))(-s_a + s_o)$. Setting $i = n_a - n_o + 1$ in Lemma 3.1, we have that

$$\begin{aligned} & P'(P^{-1}(n_a - n_o + 1))(-s_a + s_o) \\ \leq\ & (-s_a + P^{-1}(n_a - n_o + 1))P'(P^{-1}(n_a - n_o + 1)) \\ & \quad +P(s_o) - n_a + n_o - 1 \\ \leq\ & P(s_o) - n_a + n_o, \end{aligned}$$

where the last inequality follows from $s_a = P^{-1}(n_a + 1) \geq P^{-1}(n_a - n_o + 1))$. Thus, $\frac{d}{dt}\Phi \leq 3P(s_o) - 3n_a + 3n_o$ in both cases. It follows that

$$\begin{aligned} & n_a + P(s_a) + \frac{d}{dt}\Phi \\ \leq\ & n_a + n_a + 1 + 3P(s_o) - 3n_a + 3n_o \\ \leq\ & 3(P(s_o) + n_o). \end{aligned}$$

So Equation 3.2 is true if $n_o < n_a$.

Case $n_o = n_a$: We show that $d\Phi \leq 0$ or $d\Phi \leq 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$, as follows. Again, we consider 3 subcases depending whether $q_a < q_o$, $q_a = q_o$, or $q_a > q_o$.

1. Assume $q_a < q_o$. Then $n(q)$ remains zero for $q \in [q_a - s_a dt, q_a]$ and $q \in [q_o - s_o dt, q_o]$. Thus, $n(q)$ is not changed for any q and $\frac{d}{dt}\Phi = 0$.

2. Assume $q_a = q_o$. If $s_a \geq s_o$, $n(q)$ remains zero for $q \in [q_a - s_a dt, q_a]$ (which also contains $[q_o - s_o dt, q_o]$). Thus, $n(q)$ is unchanged for any q and $\frac{d}{dt}\Phi = 0$

   Else if $s_a < s_o$, $n(q)$ increases by 1 for $q \in [q_a - s_o dt, q_a - s_a dt]$. The change in Φ is $3(f(n_a(q_a) - n_o(q_o) + 1) - f(n_a(q_a) - n_o(q_o)))(-s_a + s_o)dt = 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$

3. If $q_a > q_o$, it is identical to subcase 3 of the case $n_o < n_a$, so $d\Phi \leq 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$.

It implies that $\frac{d}{dt}\Phi \leq 0$ or $\frac{d}{dt}\Phi \leq 3\Delta(n_a - n_o + 1)(-s_a + s_o)dt$. Similar argument as before shows that Equation 3.2 is true if $n_o = n_a$.

This completes the analysis when P satisfies the special conditions that we imposed at the start of this section.

3.1 Removing the special conditions on P

Consider an arbitrary power function P in the general model. We explain how to modify P so that it meets the special conditions without significantly raising the competitive ratio. If P is not increasing, one can make P undefined on those speeds where there is a greater speed

that consumes less power. Similarly, we can make P convex by eliminating any points in a subinterval (a, b) of speeds where the line segment between $(a, P(a))$ and $(b, P(b))$ lies below the curve P. We now argue that the online algorithm A with the new power function can simulate running a speed $s \in (a, b)$ in the old power function. Algorithm A can simulate any speed $s \in (a, b)$ by alternately running at speeds a and b. Specifically, assume $s = pa + (1-p)b$ for some $0 < p < 1$, we can run at speed a for a p fraction of the time and run at speed b for a $(1-p)$ fraction of the time. It gives an effective speed of s and the energy usage is $pP(a)+(1-p)P(b)$. If the slowest speed s_0 on which P is defined is not 0, then can shift $P(s)$ down by redefining P as $P(s) = P(s) - P(s_0)$, and $P(s) = 0$ for $s \in [0, s_0]$. Then it is easy to see that if A is c-competitive with respect to this new P then A is also be c-competitive with respect to the original P. If P is defined at a and b but not at any point in the interval (a, b), then interpolate P linearly between a and b. Once again the online algorithm can simulate running a speed $s \in (a, b)$ in the new power function by alternately running at speeds a and b. P stays convex by the convexity of the original P. One can find a power function between P and $P + \max\{\epsilon, \epsilon P\}$ that is defined on the same domain of speeds, is continuous, differentiable, strictly increasing, and strictly convex. The competitive ratio of A for the old power function will be within ϵ of the competitive ratio of A for the new power function. If P is bounded, where T is the maximum speed, then the natural interpretation of the algorithm is to run at speed $\min(P^{-1}(n_a + 1), T)$. We can follow the line of reasoning from [11]. We first note that at all times it must be the case that $n_a + 1 - n_o \leq P(T)$. Then the analysis essentially then goes through as in the unbounded case. In particular, in the case that $n_o < n_a$, we arrive at the same inequality of $\frac{d\Phi}{dt} \leq 3(-s_a + P^{-1}(n_a - n_o + 1))P'(P^{-1}(n_a - n_o + 1)) - n_a + n_o - 1$. If $s_a = P^{-1}(n_a + 1)$, then we have that $s_a \geq P^{-1}(n_a - n_o + 1)$ as before. Else if $s_a = T$, we still have that $s_a \geq P^{-1}(n_a - n_o + 1)$.

4 Fractional Weighted Flow Plus Energy

Our goal in this section is to prove Theorem 1.2, that in the general model, HDF plus the natural speed scaling algorithm is $(2+\epsilon)$-competitive for the objective of fractional weighted flow plus energy. We start by showing, using an amortized local competitiveness argument, that this algorithm is 2-competitive if P also satisfies the special conditions from the last section.

Definition of Online Algorithm A: The algorithm always runs the job of highest density. The density of a job is its weight divided by its work. The speed s_a^t at time t is

$$s_a^t = P^{-1}(w_a^t)$$

where w_a^t is the total fractional weight of all unfinished jobs for A at time t.

Definition of the potential function Γ. Let OPT be the offline adversary that minimizes fractional weighted flow plus energy. At any time t, let w_o^t be the total fractional weight of all unfinished jobs in OPT. For a job j, its inverse density is defined to be the ratio of its original size divided by its original weight. At any time t, let $w_a^t(m)$ denote the total fractional weight of all unfinished jobs with inverse density at least m in A at time t. Define $w_o^t(m)$ similarly for that of OPT. Let $w^t(m) = \max\{0, w_a^t(m) - w_o^t(m)\}$. We define

$$\Gamma(t) = 2\int_0^\infty h(w^t(m))dm$$

where h is the function such that for any real $w \geq 0$,

$$\frac{d}{dw}h(w) = P'(P^{-1}(w))$$

Since both $P'(x)$ and P^{-1} are increasing, $P'(P^{-1}(w)))$ is an increasing of function in w. Furthermore, $h(w)$ can be written as $h(w) = \int_0^w P'(P^{-1}(y))dy$.

We now start the amortized local competitiveness analysis. For the boundary condition, we observe that before any job is released and after all jobs are completed, $w^t(m) = 0$ for all m, so $\Gamma = 0$. For the general condition, when a job is released, $w^t(m)$ is unchanged for all m, so Γ is unchanged. When a job is processed by A or OPT, the fractional weight of the job decreases continuously to zero, so Γ is continuous and does not decrease due to the completion of a job. It remains to show that at any time t during a time interval without job arrival or completion,

$$w_a^t + P(s_a^t) + \frac{d}{dt}\Gamma(t) \leq 2(w_o^t + P(s_o^t)) \tag{4.3}$$

The rest of this proof considers any such time t and proves Equation 4.3. We omit t from the superscript and the parameter for convenience. Then,

$$\begin{aligned}
\frac{d}{dt}\Gamma &= 2\int_0^\infty \frac{d}{dt}h(w(m))dm \\
&= 2\int_0^\infty \frac{d}{d(w(m))}h(w(m))(\frac{d}{dt}w(m))dm \\
&= 2\int_0^\infty P'(P^{-1}(w(m))(\frac{d}{dt}w(m))dm,
\end{aligned}$$

where the second equality follows from chain rule.

Let m_a and m_o denote the minimum inverse density of an unfinished job in A and OPT, respectively. (Let

m_a (resp., m_o) be ∞ if A (resp., OPT) has no unfinished job.) Note that $1/m_a$ and $1/m_o$ are the density of the highest density job in A and OPT, respectively. Since A is running HDF at speed s_a, $w_a(m)$ is decreasing at the rate of (s_a/m_a) for all $m \in [0, m_a]$ and $w_a(m)$ remains unchanged for $m > m_a$. Similarly, $w_o(m)$ is decreasing at the rate of (s_o/m_o) for $m \in [0, m_o]$ and remains unchanged for $m > m_o$. We consider three cases depending on $w_o > w_a$, $w_o < w_a$ or $w_o = w_a$.

Case $w_o > w_a$: We show that $\frac{d}{dt}\Gamma \leq 0$ in this case. Note that for any $m \in [0, m_o]$, $w_o(m) = w_o > w_a \geq w_a(m)$. Thus, for any $m \in [0, m_o]$, $w(m) = \max\{0, w_a(m) - w_o(m)\}$ remains zero and does not change. It means that $\frac{d}{dt}w(m) = 0$ for $m \leq m_o$. For any $m > m_o$, $w_o(m)$ remains unchanged, so $\frac{d}{dt}w(m) \leq 0$ for $m > m_o$. Therefore, $\frac{d}{dt}\Gamma = 2\int_0^\infty P'(P^{-1}(w(m)))(\frac{d}{dt}w(m))dm \leq 0$. We have $w_a + P(s_a) + \frac{d}{dt}\Gamma \leq 2w_a < 2(w_o + P(s_o))$, so Equation 4.3 is true.

Case $w_o < w_a$: The decrease in $w_a(m)$ causes a decrease in Γ and the decrease in $w_o(m)$ causes an increase in Γ. We analyze these two effects separately and bound them appropriately. For any $m \in [0, m_a]$, $w_a(m)$ is decreasing at a rate of (s_a/m_a), which causes Γ to decrease at the rate of $2\int_0^{m_a} P'(P^{-1}(w(m)))(-\frac{s_a}{m_a})dm$. For $m \in [0, m_a]$, $w(m) = w_a(m) - w_o(m) \geq w_a - w_o$, so $P'(P^{-1}(w(m))) \geq P'(P^{-1}(w_a - w_o))$. Thus,

$$\begin{aligned} & 2\int_0^{m_a} P'(P^{-1}(w(m)))(-\frac{s_a}{m_a})dm \\ \leq\ & 2\int_0^{m_a} P'(P^{-1}(w_a - w_o)))(-\frac{s_a}{m_a})dm \\ =\ & -2P'(P^{-1}(w_a - w_o)))s_a \end{aligned}$$

For $m \in [0, m_o]$, $w_o(m)$ is decreasing at a rate of (s_o/m_o), which causes Γ to increase at a rate at most $2\int_0^{m_o} P'(P^{-1}(w(m)))(\frac{s_o}{m_o})dm$. For $m \in [0, m_o]$, $w(m) = \max\{0, w_a(m) - w_o(m)\} \leq w_a - w_o$, so $P'(P^{-1}(w(m))) \leq P'(P^{-1}(w_a - w_o))$. Thus,

$$\begin{aligned} & 2\int_0^{m_o} P'(P^{-1}(w(m)))(\frac{s_o}{m_o})dm \\ \leq\ & 2\int_0^{m_o} P'(P^{-1}(w_a - w_o)))(\frac{s_o}{m_o})dm \\ =\ & 2P'(P^{-1}(w_a - w_o)))s_o \end{aligned}$$

Summing the above two terms, we have $\frac{d}{dt}\Gamma \leq 2P'(P^{-1}(w_a - w_o)(-s_a + s_o)$. By Lemma 3.1, it is at most $2(-s_a + P^{-1}(w_a - w_o))P'(P^{-1}(w_a - w_o)) + 2(P(s_o) - (w_a - w_o))$. Since $s_a = P^{-1}(w_a) \geq P^{-1}(w_a - w_o)$, $\frac{d}{dt}\Gamma \leq 2(P(s_o) - w_a + w_o)$. We have $w_a + P(s_a) + \frac{d}{dt}\Gamma \leq 2w_a + 2(P(s_o) - w_a + w_o) \leq 2(P(s_o) + w_o)$, hence proving Equation 4.3.

Case $w_o = w_a$: For $m \in [0, m_o]$, $w_o(m)$ is decreasing at a rate of (s_o/m_o), which causes Γ to increase at a rate at most $2\int_0^{m_o} P'(P^{-1}(w(m)))(\frac{s_o}{m_o})dm$. For $m \in [0, m_o]$, $w(m) = \max\{0, w_a(m) - w_o(m)\} \leq w_a - w_o = 0$, so $P'(P^{-1}(w(m))) = P'(P^{-1}(0))$ and $\int_0^{m_o} P'(P^{-1}(w(m)))(\frac{s_o}{m_o})dm = P'(P^{-1}(0)))s_o$. By Lemma 3.1 with $i = 0$, $s_a = 0$, we obtain that $\frac{d}{dt}\Gamma \leq 2P'(P^{-1}(0)))s_o \leq 2P(s_o)$. We have $w_a + P(s_a) + \frac{d}{dt}\Gamma \leq 2w_a + 2P(s_o) = 2(w_o + P(s_o))$, hence proving Equation 4.3.

This completes the analysis when the special conditions hold. The proof can be extended to the general model in the same way as was done in the proof of Theorem 1.1.

References

[1] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. In *Lecture Notes in Computer Science (STACS)*, volume 3884, pages 621 – 633, 2006.

[2] N. Bansal, H.L. Chan, T.W. Lam, and L.K. Lee. Scheduling for bounded speed processors. In *International Colloquium on Automata, Languages and Programming, ICALP*, 2008, to appear.

[3] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. submitted to SODA 2009.

[4] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 805–813, 2007.

[5] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Kirk R. Pruhs. Online weighted flow time and deadline scheduling. In *Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 36–47, 2001.

[6] Fred Bower, Daniel Sorin, and Landon Cox. The impact of dynamically heterogeneous multicore processors on thread scheduling. Unpublished.

[7] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[8] J. Adam Butts and Gurindar S. Sohi. A static power model for architects. In *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, pages 191–201, 2000.

[9] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, 2007.

[10] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, 1952.

[11] T.W. Lam, L.K. Lee, Isaac To, and P. Wong. Speed scaling functions based for flow time scheduling based on active job count. In *Proc. of European Symposium on Algorithms, ESA*, 2008, to appear.

[12] John Markoff and Steve Lohr. Intel's huge bet turns iffy. New York Times, September 29 2002.

[13] Ke Meng and Russ Joseph. Process variation aware cache leakage management. In *ISLPED '06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 262–267, 2006.

[14] Kirk Pruhs. Competitive online scheduling for server systems. *SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[15] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.